

MicVO -lectures

~~without~~  $\Rightarrow$  418 to 470 Jip G

## Lec. slide 1. Micro

What is Embedded system?

↳ Combination of hardware and software.

↳ Designed to perform specific function.

↳ combination of computer hardware with the software to control, monitor, communicate to do multiple operations

ex: washing machine, Digital camera, DVD Player.

General Purpose system vs. Embedded system

General Purpose	Embedded.
↳ combination of <del>any</del> <sup>generic</sup> HW & general purpose OS for executing variety of apps.	↳ combination of special purpose <del>OS</del> HW and Embedded OS for executing specific set of apps.
Need not to be deterministic in execution behavior	Execution behavior is deter. for certain types of embedded system.
Example: Computer, laptop	Example: DVD, Printers.

## Characteristics of Embedded systems:

- \* Single Functioned.
- \* Reliability.
- \* Cost effectiveness
- \* low power Consumption
- \* Fast execution time
- \* efficient use of memory.
- \* Processing Power.

## \* Major application areas of Embedded system:-

- 1) Consumer electronics (Cameras, etc)
- 2) Household appliances (TV, DVD Players, Washing machine)
- 3) Home automation & security system (Fire alarms - etc)
- 4) Telecom (handset, Cellular telephones - etc)
- 5) Healthcare (scanners, ECG, etc)
- 6) Banking & retail (ATM, currency counter, etc)

## Categorization of embedded system

### [1] Based on Generalization:-

a. First generation (from <sup>4</sup>8 to 8 bit microcontrollers & ex: stepper motor control units)

b. 2nd generation (8 to 16 bit micro controllers & ex: sensors)

c. 3rd " (16 to 32 bit & ex: DSP)

d. 4th " (Smart phones, Mobile internet devices, etc)

[2]



[2] Based in complexity & Performance :-

a. Small scale Embedded system

↳ small Processing elements with simple Program

↳ low cost.                      ↳ lower Performance.

b. Medium scale ES :-

↳ slightly complex in HW & SW.

↳ low cost.                      ↳ Medium Performance.

c. Large scale ES

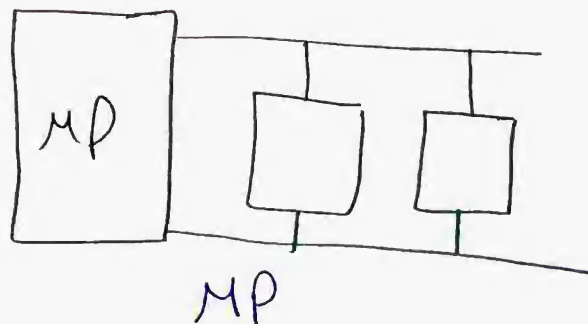
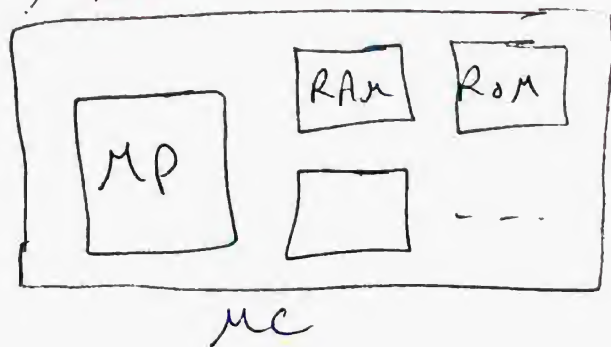
↳ complex                      ↳ High Performance.                      ↳ low cost

[3] Based on deterministic behavior.

↳ Hard real time                      ↳ soft real time.

[A] Based on Triggers (Event based & time based)

Micro Controller vs Micro Processor



~~PC~~ ES → uses (MC) to do specific task.

PC → uses MP to do different things.

[3]

## \* Criteria of choosing MC

- \* Speed
- \* Power Consumption
- \* Cost
- \* RAM & ROM
- \* I/O Ports
- \* Packaging (Put MC in which Package)
- \* Community

## Components of 8051 MC

- a) 128 bytes RAM
- b) 4K ROM
- c) 2 timers
- d) one serial Port
- e) 6 interrupt
- f) 4 I/O Ports (each 8 bits)

## Notes

- 1) 8051 is subset of 8052
- 2) 8031 " a ROM-less 8051

	8051	8052	8031
ROM	4K	8K	0K
RAM	128	256	128
Timers	2	3	2
I/O Pins	32	32	32
Serial Port	1	1	1
Interrupt Sources	6	8	6

- 3) \* vast majority of 8051 register are 8-bit registers.

## Micro Processor V-s Micro Controller

Micro Processor	Micro Controller
→ use CISC Architecture	→ use RISC and Harvard.
→ Has ROM, RAM, secondary storage memory, I/O etc Placed on board and Connected through Buses.	→ ROM, RAM, I/O PrePherals are Combined in single integrated circuit
→ Less secured	→ more secured.
→ expensive	→ cheaper
→ Design takes more time	→ Design it takes <del>less</del> less time.

RISC	CISC
simple instruction Format	variable inst. Format
large set of CPU registers	small set of general Purpose registers
very few addressing modes	large number of addressing modes.
simple Pipelining	Complex Pipelining
supports on chip cache memory	seldom supports on chip cache memory



## 8051 Assembly language Programming

1) How instructions are executed.

2) Registers.

- a) 8 bit registers Me
- b) A: Accumulator, must be destination of any operation.
- c) B, PC, DPTR, SP
- $R_0 - 7$

Any Assembly Program { instructions & directives }

instructions  $\Rightarrow$  [label:] opcode [operands] [; Comments]

Directives (not instructions)

↳ but they help assembler in assembly Process.

[ex] ORG address  $\Rightarrow$  PC = address

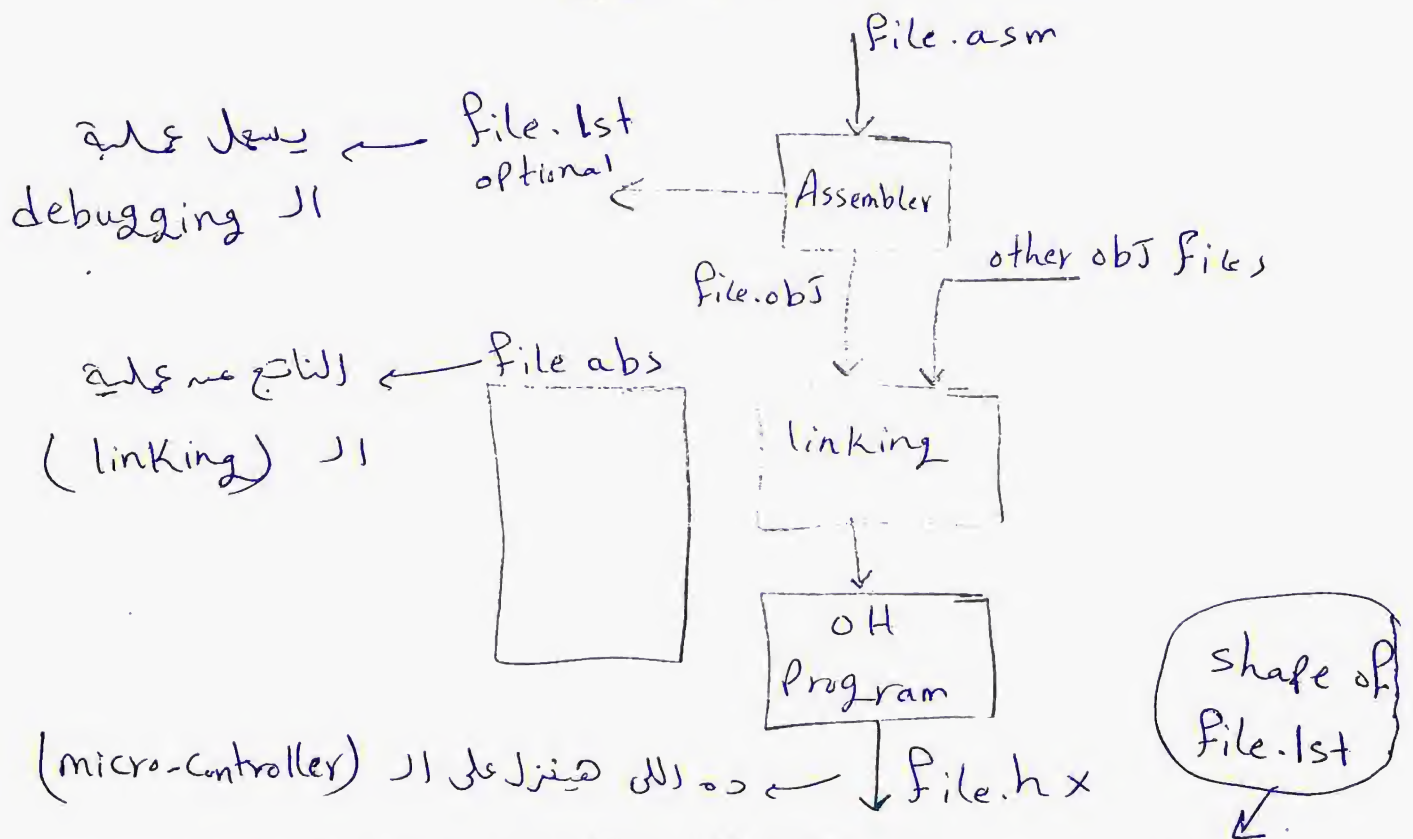
له معناها انه البرنامج بتاعي اول (instruction) هيقع موجود في ال (Address) اللي عنده ده وهو اللي ال (Program Counter) هيشير عليه في الاول.

→ END

يعرف ال (Assembler) انه دي نهاية الجزء اللي بيعمل ليه (Assembly).

## step of assembly language Program

(File.asm) يكتب فيه ار (instruction) الى عايزها  
 له بيخد على ار (assembler) يطلع (file.object)  
 وده التالى الى انتا محتاج يحصل له (linking) مع  
 الحاجة الى انتا عايز تعلقها .



Address	Machine	Assembly	Comments
0000	7D25	ORG 0000	
0002	7F34	MOV R5, #25	
0004	2D	MOV R6, #34	
0005	—	ADD A, R5	



من أول ما تفتح الـ (Program) هتلاقى لـ

الـ PC عندك فيه  $PC = 0000$  وكل

شوية عمل (increment) وهيقارن

لـ الـ الـ جاي مش (instruction)

0000	7D
0001	25
0002	7F
0003	34
:	:

RAM

ملحوظة: المقدرة أكتب برنامج السعة بتاعته أكثر من

(4K byte) لأن سعة الـ RAM ← 4K byte

(2) هيدل برنامج (assembly) ديطلب سلة تصلا الجدول

بتاع الـ (lst. file) في الصفحة السابقة وبعدين تعمل الجدول في

الـ الصفحة دي رى مثال في م 75/74 في (slides)

### DB directive

↳ most widely used ↳ used to define 8-bit data

شركة لو عاين أطبع (Hello world) بار (assembly)

لـ محتاج تكتب موجود في الـ (memory)

ORG 500

Data 1 DB "Hello world"

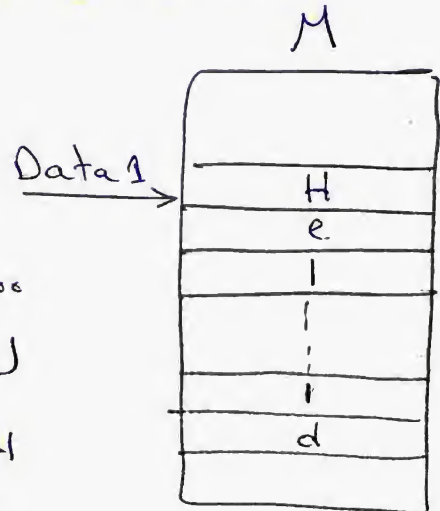
لـ عند الـ (label Data 1) الـ هو عبارة عن

500 كأي يقول إنه بيشار على الأمر 500.

لـ أنا كده عرفت بار (DB) مجموعة الأوامر

الموجودة من أول الـ (address 500)

محمولة لـ (Hello world)



## \* EQU

له بديل (define) لثابت من غير ما يلاحظ مكانه في ال (memory)  
 له بديل في ال (data label) وادلما ال (label)  
 يظهر في البرنامج يستخدم قيمة الثابت خوراً.

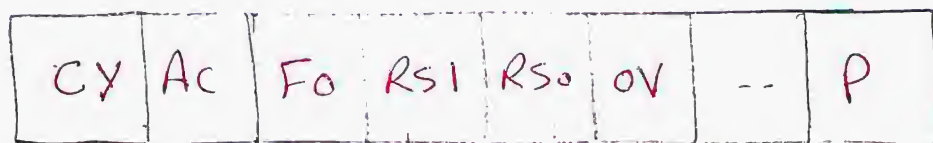
ex

~~Count~~ EQU 25 → use EQU For Counter Constant

Mov R3, #Count → Constant used to load R3.

Flag register (PSW) → Program status world

لغ عبارة عن (8-bit register) يستخدم 6-bit



↳ not used

Where

CY → Carry. Ac → Auxiliary carry.

RS0, RS1 → Bank selectors. OV → overflow.

P → Parity flag. Fo → user definable.

ex

$$\begin{array}{r}
 00111000 \\
 00101111 \\
 \hline
 01100111 \\
 \hline
 6 \quad 7
 \end{array}$$

$$\begin{array}{r}
 38 \\
 + 2F \\
 \hline
 67
 \end{array}$$

P=1, Ac=1, CY=0

عدد البتات  
 P=1 ← odd  
 P=0 ← even  
 CY=0 ← 0=D7  
 إنتقلت من D3 ← D4 ومعاله  
 Ac=1 ← carry

\* RAM : 128 bytes

له مكان اخذ و فني ال (Variables)

اللي بيجل فني (operations)

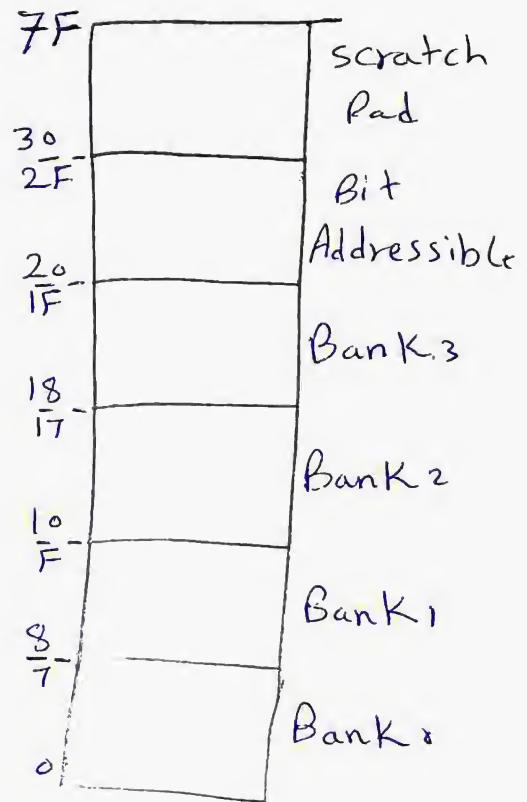
↳ (RS1, RS0)

حسب قيمه بختار ال (Bank)

اللي ال (register) بتاعي بيتعامل

معاه .

RS0	RS1	Bank
0	0	0
0	1	1
1	0	2
1	1	3



(stack) (default) بتاعه ده ال (Bank 1)

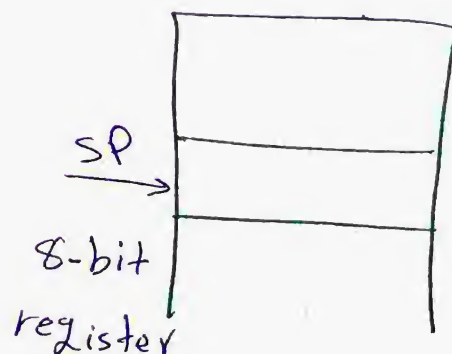
↳ push & pop

Default

SP = 7

عنه ال (Bank 1) بتاعي

(8) نه





Push → storing CPU register in stack  
 → increment then push.

Pop → loading contents of stack back into  
 CPU register.  
 → pop then decrement.

ملحوظة: لا يشترط أن  $(SP=7)$  هو فقط بيشار على  
 عنوان معين ويمكن العنوان الذي بعده هو الذي يتخزن فيه  
 الـ (data) باعتبار.

**[Ex]** show stack and stack pointer

Mov R6, #25H

Mov R1, #12H

Mov R4, #0F3H

Push 6

Push 1

Push 4

**[Sol]**

0B	
0A	
09	
08	

SP=7

after Push 6

0B	
0A	
09	
08	25

SP=08

Push 1

0B	
0A	
09	12
08	25

SP=09

Push 4

0B	
0A	0F3H
09	12
08	25

SP=0A

**[u]**

## JUMP, Loop and CALL instructions

Page 104

Sec 6

Loop repeat sequence of instructions a certain number of times.

is performed by  $\Rightarrow$  DJNZ Reg, label

بعد (decrement) لا (reg) و (check) كل شوية  
، طالما ليس بهنجر يعمل (Jump) لا (label)

ex Reg = value

بعد (decrement) لا (reg) وطول

ما هو ليس بهنجر يعمل (loop)

DJNZ Reg, loop

ماذا لو هحتاج أكرر (action) أكثر من 256 مرة

له أكبر قيمة لا (reg) هي 255 (FF) لكنه لو عندي (loop)

تحتاج قيمة أكبر من (255) فعل أكثر من (loop) جمعه  
يعني (nested loop)

لم يعني مثلاً لو عندي 700 فعل (2 loops) واحدة 10  
والأخرى 70 ، والناتج يكون حاصل ضربهم

## Jump

### Conditional Jump:

لـ بعد (operation) تم العمل (check flag) وعلى أساسه يعرف أخذ أي (action) في البرنامج.

ex JZ label

لـ يعمل (check) على (zero flag) ومنها العمل (Jump) لـ (label)

if equal to zero  $\Rightarrow$  jump to label.

else

(unconditional) في المنطقة \* عمل (label 2) و (Jump)

ex  
JZ label 1  
\*  
label 1  
label 2

## Jump

### 1) LJMP (long Jump)

لـ بعد (Jump) لأي مكان في البرنامج.

لـ رتبة عبارة عن (3 bytes) واحدة لـ (opcode) ،  
اثنين لـ (address)

### 2) SJMP (Short Jump) (-128 to 127)

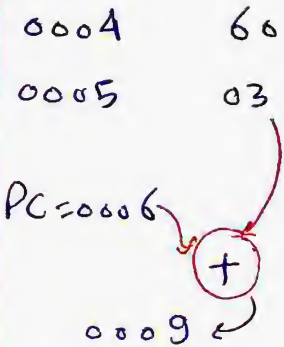
لـ بقدر العمل (Jump) فوق ارتحت بعد أقصر (128 bit)  
لـ عبارة عن (2-byte)



# Page 112 ex

Line	Pc	oPCode	operand
01	0000		ORG 0000
02	0000	7800	MOV R0, #0
03	0002	7455	MOV <del>R0</del> , #55H
04	0004	6003	JZ NEXT
05	0006	08	INC R0
06	0007	04	AGAIN: INCA
07	0008	04	INCA
08	0009	2477	NEXT: Add A, #77H
09	000B	5005	JNC OVER
10	000D	E4	CLR A
11	000E	F8	MOV R0, A
12	000F	F9	MOV R1, A
13	0010	FA	MOV R2, A
14	0011	FB	MOV R3, A
15	0012	2B	OVER: ADD A, R3
16	0013	50F2	JNC AGAIN
17	0015	80FE	HERE: SJMP HERE
18	0017		END

ex



صاحب جیب (oPCode)  
لا loop او jump

عند JNC OVER  
عائز يطلع ب 5005  
قبيط طرح (0012) مكان  
ما ار (OVER) راحت  
من ار (0D) ار PC

كذلك مع  
JNC AGAIN

لوار (oPCode)  
ممكن من رقمين ار  
PC هيزيد ب (2)  
ولو رقم واحد ار  
(PC) هيزيد (1)

## Call

هو يكتب شوية أكواد وفي النيه عايز أعمل  
(subroutine) فيعمل ليه (Call)

\* لا يتعمل (Jump) بتكمل الكود بتاعك  
لكه مش بترجع للبداية .

لكه في ال (Call) بتستخدم أمر  
(return) عشان ترجع للبداية .

ex

====

Call label1

====

Sub1:

====

RET return

هو ال (Call) بيعمل Push لا (PC) ويعمل تغيير ليه  
ر (Sub1) عشان يشاور عليه

Push PC

change PC to sub1

Sub1:

====

RET

=> POP PC

قبل الجزء بتاع (Sub1)

صونغ (HLT) صغناها

نهاية البرنامج

## Call

له يعمل (Call) في أي مساحة في البرنامج Long call (Lcall)

له ال (subroutine) موجود في أي مكان  
خلاف ال (64Kbyte) .

له يعتبر (3-byte instruction)

2) Acall (absolute call)

له يعمل (Call) في صدد (2K-byte) فقط

له (2-byte instruction)

له (2-byte instruction)



## Notes

له لو في ار (main) دار (Subroutine) بتستعمل نفس

ار (registers) ← ار (Call) مش معن بانه يعنى ار

ار " لأنه بيتعامل مع ار (Push, return) فقط.

ل في هذه الحالة قبل بداية ار (Subroutine) عمل Push

لقيم ار (registers) اللى هستخدمها وفي الآخر عمل (Pop) ليهن  
بس بالعكس.

له لو علت (Push) ودمفيسر في الآخر (Pop)

مش صيالح قيم ار (PC) المطلوبة.

له لازم كل (Push) في البداية يقابلها

نفس ار (Pop)

Push 4

Push 5

Pop 5

Pop 4

\* ار (Instruction) يتم تفرقة بال (Crystal oscillator)

ار machine cycle : هي (cycle) يتم تنفيذ الامر فيها

له ار (microcontroller) يحتوى على مجموعة من ار (operations)

كل (operation) تحتاج كام (cycle)

ار (machine cycle) ل (atmel) عبارة عن (12 clocks)

→ Find Period of machine cycle for 11.0592 MHz  
crystal frequency.

$$\frac{11.0592}{12} = 921.6 \text{ KHz} \quad \& \quad \text{machine cycle} = \frac{1}{921.6 \text{ KHz}} = 1.085 \mu\text{s}$$

16



## Addressing modes

1) Immediate

2) register.

3) Direct

4) register indirect

5) Indexed.

↑  
↑  
↑  
Accessing memories.

### 1) Immediate

↳ src. operand is constant. `MOV A, #25H`

↳ load info. into any registers, including 16-bit DPTR register.

### 2) Register

↳ hold data to be manipulated (`MOV A, R0`)

↳ src. & dst. registers must match in size

ex `MOV DPTR, A (xx)` & `MOV DPTR, #25F5H`

↳ `MOV R4, R7 (xx)`

### 3) Direct addressing mode

↳ used to access RAM locations 30-7FH

ex: `MOV A, 4` & `MOV A, R4`

no # sign here

\* SFR (special function register)

↳ can be accessed by their names or their addresses

↳ have addresses

between 80H

and FFH.

`MOV 0E0H, #55H`

`MOV A, #55H`

`MOV 0F0H, R0`

`MOV B, R0`

**[Ex]** write code to send 55H to ports P1 & P2 using.

a) their names

Mov A, #55H

Mov P1, A

Mov P2, A

b) their addresses

Mov A, #55H

Mov 80H, A

Mov 0A0H, A

---

\* Stack and direct addressing mode

↳ only direct addressing mode is allowed for pushing or popping the stack.

↳ Push A (invalid)  $\Rightarrow$  it should be Push 0E0H

**[Ex]** show the code to push R5 and A onto the stack and then pop them back into R2 & B, (B=A, R2=R5)

**[sol]**

Push 05 ; Push R5 to stack

Push 0E0H ; Push register A to " .

Pop 0F0H ; Pop top of stack to B  
now B = A

Pop 02 ; Pop top of stack into R2  
now R2 = R5

#### [4] register indirect addressing mode:-

↳ register is used as a pointer to data (only R0, R1)

MOV A, @R0 → move contents of RAM whose address is held by R0 into A

MOV @R1, B

[EX] write a program to copy value 55H into RAM memory locations 40H to 41H using:

a) direct addressing mode

MOV A, #55H

MOV 40H, A

MOV 41H, A

b) register indirect without loop

MOV A, #55H

MOV R0, #40H

MOV @R0, A

INC R0

MOV @R0, A

c) register indirect with loop

MOV A, #55H

MOV R0, #40H

MOV R2, #02

AGAIN: MOV @R0, A

INC R0

DJNZ R2, AGAIN

↳ advantage of register indirect that it makes accessing data dynamic rather than static as in direct addressing mode.